ApexClaw: A Persistent Infrastructure for Agentic Scientific Research

Zhisheng Tang and Mayank Kejriwal *

Posted Date: 19 May 2026

doi: 10.20944/preprints202605.1178.v1

Keywords: AI scientists; infrastructure; metascience; OpenClaw; agents-for-science

 doi:10.20944/preprints202605.1178.v1

Article

ApexClaw: A Persistent Infrastructure for Agentic Scientific Research

Zhisheng Tang and Mayank Kejriwal *

GRAIL
* Correspondence: mayank@grail.page

Abstract

Most AI-for-science systems (agents4science) are evaluated as task-specific automation rather than persistent work environments. This leaves an important and pragmatic systems-question unresolved: what infrastructure enables scientific agents to be trustworthy, steerable, and reproducible? We present ApexClaw, a persistent workspace for human-supervised AI agents in scientific discovery. The system provisions isolated Linux environments with scientific computing libraries, a browser-based IDE, and a registry of reusable scientific skills. Rather than pursuing full autonomy, ApexClaw emphasizes human oversight through an interface that exposes agent reasoning, tool calls, and workspace files, allowing scientists to guide agents at key junctures. We validate this approach with telemetry from 81 users operating 243 workspaces across 252 conversations (March–May 2026): 87% of users reconnected to existing workspaces, and humans intervened selectively on 3.4% of interactions. These findings demonstrate that hybrid autonomy with persistent context is ordinary practice in real agentic science, and that durable workspaces, explicit artifacts, and auditable traces are necessary infrastructure for reproducible and steerable AI scientists.

Keywords: AI scientists; infrastructure; metascience; OpenClaw; agents-for-science

1. Introduction

The automation of scientific discovery has long been envisioned as a path toward accelerating research. Early symbolic systems such as DENDRAL [1] and the Automated Mathematician [2] pioneered computational exploration of scientific creativity [3–5], establishing a trajectory toward fully automated discovery [6]. Recent work by Langley [7] reinforces this vision with integrated computational systems for scientific discovery. Today, AI systems are moving from narrow tools for isolated scientific tasks toward agents that participate in broader research workflows.

This transition has become explicit at venues like the *Agents4Science* conference[1], which asks how AI systems might contribute to ideation, hypothesis generation, analysis, manuscript writing, review, and disclosure. Recent work demonstrates this ambition with striking clarity. The *AI Scientist* [8,9] represents a landmark system automating the end-to-end research cycle—from hypothesis generation and experimental design through code execution, visualization, paper writing, and simulated peer review. Similarly, *Agent Laboratory* frames LLM agents as research assistants that move from initial research ideas through literature review, experimentation, and report generation. *AI co-scientist* studies multi-agent hypothesis generation and refinement with active scientist guidance, particularly in biomedical contexts.

These systems share the vision that scientific agents are no longer merely question-answering interfaces; they are becoming participants in the production of research artifacts themselves. *ResearchAgent* [10] and *SciAgents* [11] use multi-agent reasoning and knowledge graph integration to generate novel research ideas. Modern frameworks like *AutoGen* [12] enable specialized agents to collaborate

[1] https://agents4science.stanford.edu/

 doi:10.20944/preprints202605.1178.v1

on complex tasks, with applications ranging from mechanics research [13] to autonomous chemistry [14,15]. These advances are powered by reasoning techniques such as chain-of-thought prompting [16], iterative self-reflection [17], and executable code actions [18]. More recent systems like AIDE [19] continue this trajectory, exploring structured exploration within code-level environments.

However, critical limitations remain. Recent evaluation efforts—including MLE-bench [20], a specialized benchmark for machine learning engineering tasks—reveal that while LLMs can generate ideas judged as novel [21], feasibility and practical validation remain challenging. Tools such as GraphEval [22] attempt to address the difficulty of accurately assessing automated research ideas. Recent work on chain-of-ideas approaches [23] and evidence from virtual lab studies [24] suggest agentic systems are most effective as partners to human scientists rather than as autonomous agents. This points to a deeper problem: existing agentic systems are typically evaluated as reasoning pipelines or task-specific automation rather than as persistent work environments where scientists can inspect, modify, and reuse computational artifacts.

This mismatch between agent capability and scientific practice is not just an engineering nuisance. Consider a typical workflow (Figure 1): a scientist asks an agent to perform exploratory data analysis on a novel dataset. The agent analyzes the data, generates visualizations, and returns a summary. This single-turn interaction provides little opportunity for course correction. If the agent made an erroneous statistical assumption, selected the wrong clustering method, or misinterpreted a data anomaly, the scientist may not discover it until after the interaction is complete. Even if flaws are caught, the scientist must then restart: running the analysis again with corrections, re-executing dependent steps, and piecing together what changed. Multi-turn conversations help, but they are still transactional; context is lost when the session ends, and previous scripts, logs, and intermediate results scatter across the scientist's file system [25,26]. When multiple agents collaborate on a research project, or when a project spans weeks, this fragmentation becomes critical. Teams lose track of why decisions were made, which code versions were used, and how results relate to earlier hypotheses [27,28]. The researcher's notebook — once the gold standard for recording the path from question to answer — has no digital equivalent in current agent systems.

 doi:10.20944/preprints202605.1178.v1

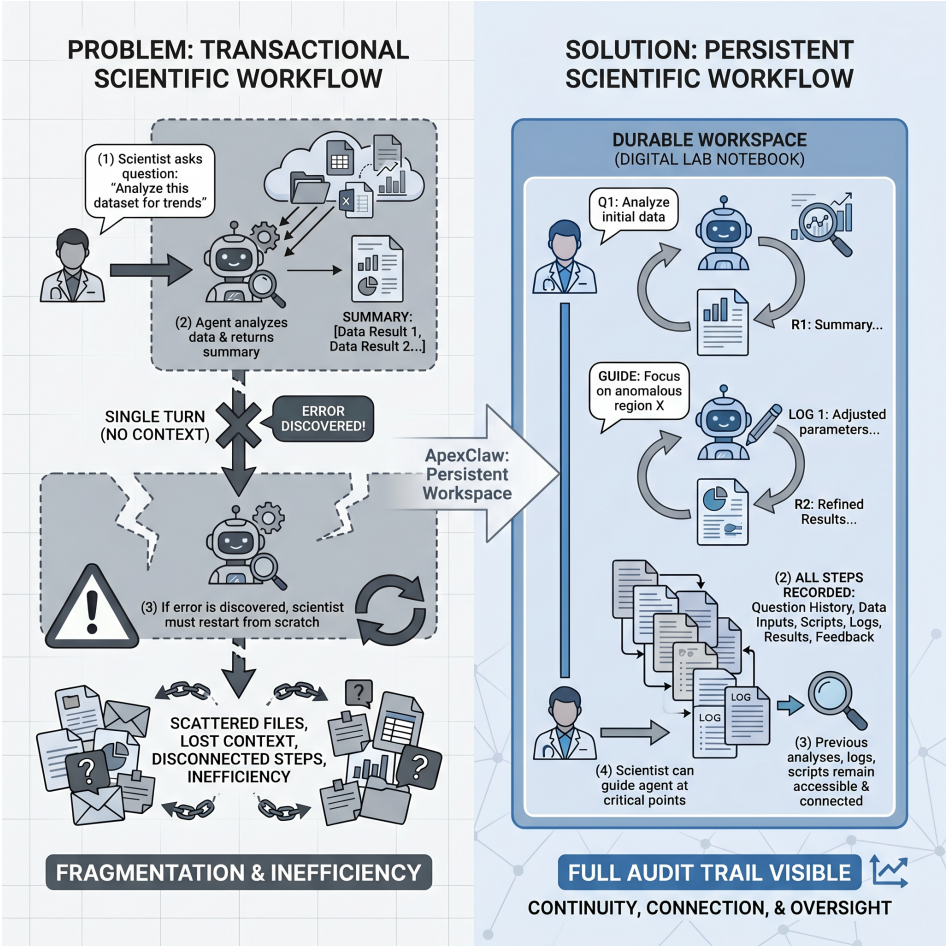

Figure 1. Transactional versus persistent scientific workflows. Current agent systems (left) handle single-turn interactions where errors trigger complete restarts, scattering artifacts and losing context. ApexClaw's persistent workspace (right) maintains continuous engagement with full audit trails, allowing scientists to guide agents at critical points and retain all intermediate analyses, logs, and decision records.

Scientific practice is fundamentally artifact-heavy and cumulative. It involves datasets, scripts, package versions, figures, intermediate files, statistical assumptions, citations, and human decisions embedded throughout [29]. When an agent produces a result, the scientist must be able to trace how that result emerged from evidence. The agent should not only produce a plausible answer but also leave a reviewable path from prompt to evidence, including tool calls, data transformations, generated visualizations, and explicit moments where the scientist provides guidance [30,31]. This creates a more accountable form of AI assistance than either a single-turn response or an opaque autonomous pipeline. Without such accountability, agent outputs risk becoming black boxes, difficult to debug, impossible to extend, and unsuitable for publication or reproducibility.

In this paper, we introduce ApexClaw, an OpenClaw-based infrastructure that addresses this gap by treating persistent workspaces as a first-class concern for agentic science [32,33]. Rather than claiming fully autonomous discovery, ApexClaw focuses on the practical conditions required for trustworthy AI-assisted science:

1. Durable workspaces where agents operate and context persists across sessions (enabling scientists to resume interrupted work and agents to build on previous computations) [34,35];
2. Explicit artifacts for scientific review and collaboration (code, data, logs, visualizations all remain in the workspace for inspection);
3. Reusable skills that encapsulate domain knowledge and can be composed across projects (reducing agent reasoning overhead while standardizing scientific procedures);

 doi:10.20944/preprints202605.1178.v1

4. Human oversight at ambiguous or high-stakes decision points (through a browser-based IDE and structured question-answer interactions); and
5. Auditable traces of reasoning, tool use, and guidance interactions (complete logs of agent work and human interventions) [36,37].

ApexClaw's positioning is complementary to existing agent systems: it provides the 'metascience' workspace substrate within which multiple agent architectures, reasoning strategies, and models can be deployed, evaluated, and directly compared on the same scientific problem within the same persistent environment.

2. System Design and Workspace Architecture

For scientists, the immediate impact of a persistent agent workspace is reduced friction across routine but time-consuming research tasks. Literature search, public database lookup, exploratory analysis, statistical testing, visualization, and report drafting often require switching between many tools (spreadsheet software, statistical packages, visualization platforms, document editors) and repeatedly reconstructing context across disconnected interfaces. ApexClaw consolidates these tasks into one workspace where the agent can coordinate tools while the scientist remains able to inspect intermediate results, correct problematic choices, and reuse successful procedures. Figure 2 illustrates the ApexClaw workspace, where agents execute code and scientists maintain continuous visibility into reasoning, tool calls, intermediate results, and resource usage.

The broader impact is methodological: by encouraging agents to produce durable artifacts rather than only final prose, ApexClaw supports a more accountable style of AI-assisted science where scientists can review generated code, rerun analyses with modified assumptions, compare alternative workflows, and extract reusable procedures for future projects.

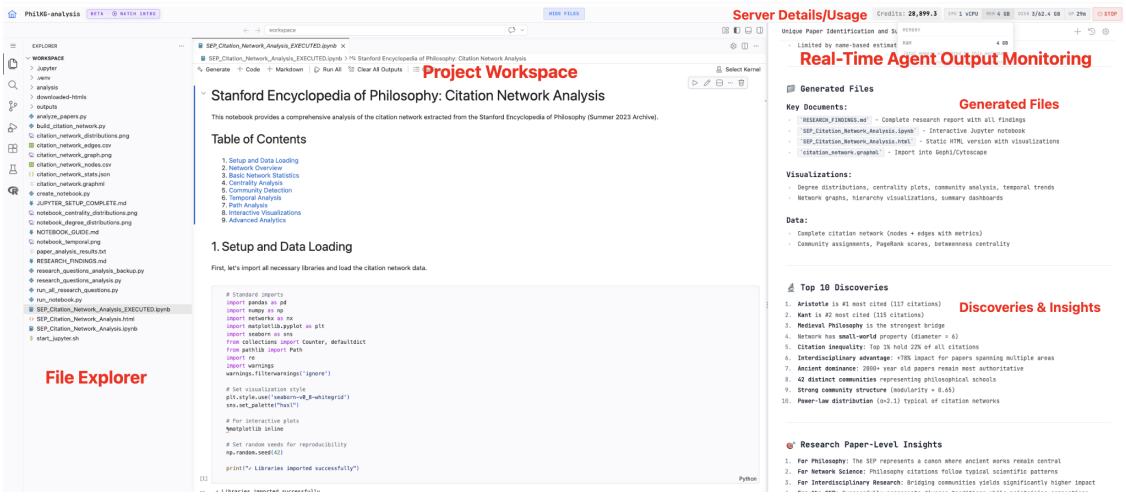

Figure 2. ApexClaw's interactive workspace interface showing the Jupyter notebook and code execution (project workspace), file explorer, and real-time agent output monitoring.

Figure 3 shows the five-layer architecture of ApexClaw. The infrastructure provisions isolated, persistent Linux workspaces where agents can execute code and scientists can inspect results. Each workspace includes a curated scientific computing stack (NumPy, SciPy, Pandas, scikit-learn, Matplotlib, SymPy, R with tidyverse, and domain-specific libraries such as Biopython and NetworkX). Files and installations persist across sessions, allowing scientists to resume interrupted work and agents to build on previous computations.

Agents are model-agnostic: different LLM providers and prompting policies can be configured through the agent runtime, enabling comparisons across model backends while preserving the same workspace environment. Agents interact with the workspace through a stable execution interface that exposes Python code execution, file operations, and event logs. This separation allows comparing agent

 doi:10.20944/preprints202605.1178.v1

behaviors across different models and reasoning strategies while holding the workspace environment constant.

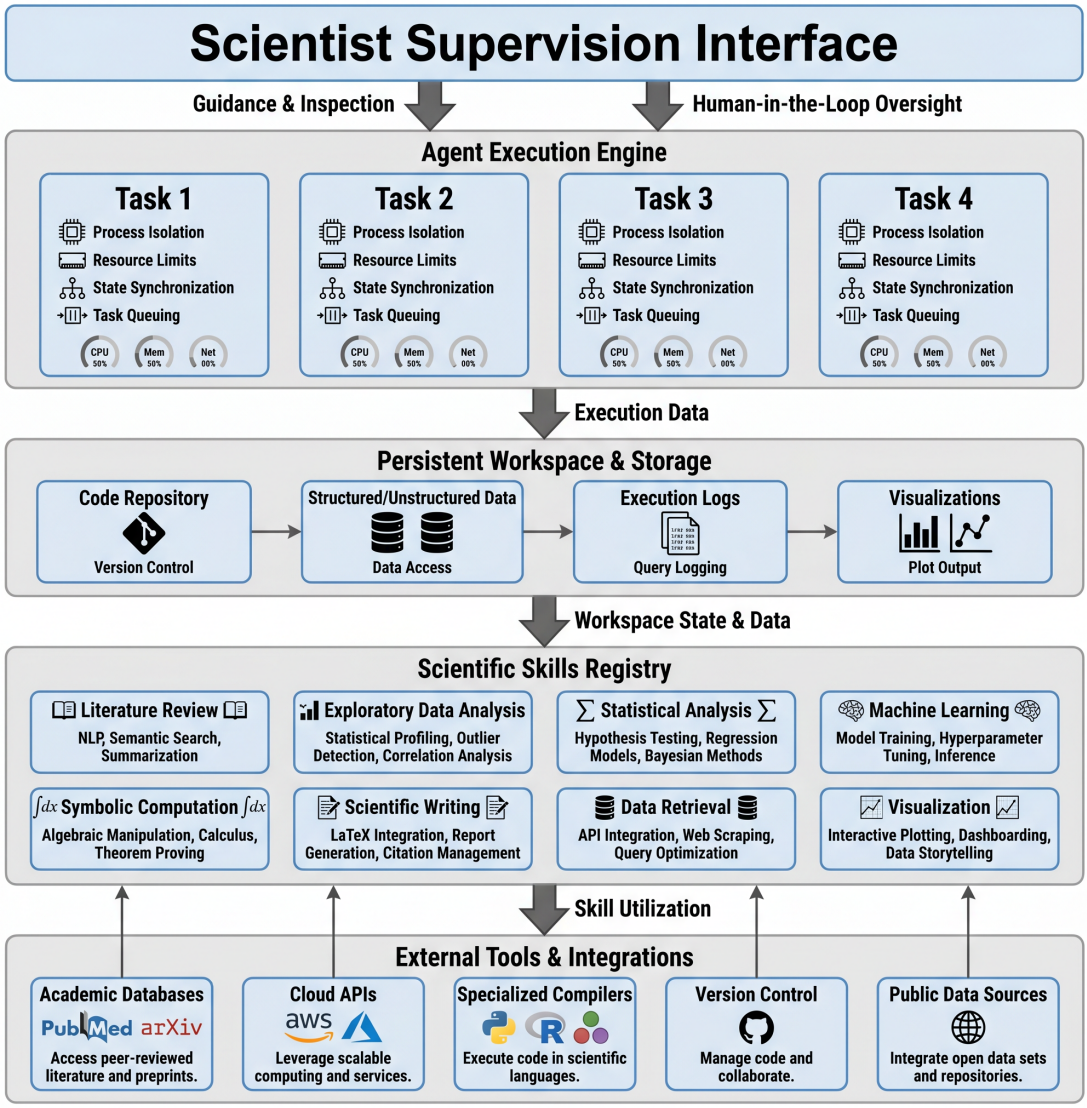

Figure 3. ApexClaw system architecture comprises five integrated layers: (1) the **Scientist Supervision Interface** providing a browser-based IDE and conversational workspace interaction, (2) the **Agent Execution Engine** managing isolated task execution and state updates, (3) the **Persistent Workspace and Storage** layer maintaining code, data, logs, and visualizations, (4) the **Scientific Skills Registry** offering reusable domain-specific procedures, and (5) **External Tools and Integrations** for database access and third-party service connectivity.

ApexClaw provides scientific *skills*—reusable procedures that encapsulate domain knowledge for common tasks (exploratory data analysis, hypothesis testing, model fitting, visualization, literature retrieval, symbolic computation). Skills reduce the burden on agents to reason about low-level library details. Scientists can define custom skills for domain-specific workflows, and agents invoke skills through workspace-local definitions or skill URLs.

2.1. System Tools and External Integrations

ApexClaw agents access a curated toolkit of system-level capabilities. For computational work, agents execute Python scripts within isolated workspaces that include NumPy, SciPy, Pandas, and Scikit-learn for numerical computation and machine learning. For statistical analysis, agents can invoke R through the command line, accessing packages like tidyverse, ggplot2, and specialized packages (DESeq2 for genomics, igraph for network analysis). Agents can also call command-line tools:

 doi:10.20944/preprints202605.1178.v1

ImageMagick for image processing, FFmpeg for video analysis, Graphviz for graph visualization, and BLAST for sequence alignment. Within Python, agents can generate publication-quality figures using Matplotlib and Seaborn, create interactive visualizations with Plotly, and produce network diagrams with NetworkX. All generated files are written to the persistent workspace and remain available for scientist review and further processing.

ApexClaw supports connections to MCP servers, allowing agents to interface with external tools and data sources through a standardized protocol. Users can configure MCP server connections as needed for their workflows. Through MCP servers, agents can access tools such as literature search (PubMed, arXiv), sequence databases (NCBI GenBank, UniProt), or public repositories (Gene Expression Omnibus, Zenodo, Kaggle), among others. Each MCP server translates agent requests into properly-formatted queries and parses responses back into structured format (e.g., JSON metadata lists or CSV tables) that agents can process further.

For molecular biology workflows, agents can call AlphaFold2 (via its Python API) to predict protein structures from amino acid sequences, storing outputs (PDB files) in the workspace. For chemistry workflows, RDKit enables molecular descriptor calculation, substructure matching, and SMILES string processing. For systems biology, agents invoke CellChat for cell-cell communication analysis, or PCSF (Prize Collecting Steiner Forest) for network inference from omics data. Each tool is installed and configured at workspace creation; agents can discover available tools through a registered tool inventory that specifies input/output formats.

Every tool invocation is logged with: the tool name and parameters, the timestamp, the agent request that triggered it, the return value or error message, and the output files or data written. This log is stored as a structured JSON file in the workspace and exposed through the supervision interface, allowing scientists to understand the complete execution trace and reproduce or modify any step. If a tool call fails (e.g., a network error during database download), the agent observes the error and can decide whether to retry, use cached data, or try an alternative approach. Scientists can inspect failure modes and guide the agent toward more robust strategies.

2.2. Scientific Skills: Reusable Domain Procedures

ApexClaw's *Scientific Skills* layer encapsulates common domain knowledge into reusable procedures that reduce agent reasoning overhead and standardize scientific practice. A skill is a parameterized workflow—a Python or R script with a well-defined interface (inputs and outputs) that solves a recurring scientific task. When an agent invokes a skill, ApexClaw binds the requested parameters, executes the script in the workspace, and returns structured results.

Data Analysis skills include: the Exploratory Data Analysis (EDA) skill accepts a CSV or Parquet file, automatically detects data types, generates summary statistics (mean, median, std, quartiles, missing values per column), creates histograms for numeric columns, produces cross-tabulation tables for categorical columns, and saves a formatted HTML report. A Hypothesis Testing skill implements common statistical tests (t-test, ANOVA, chi-squared, Mann-Whitney U) given a data matrix and group labels, returning test statistics, p-values, and effect sizes with interpretation. A Time Series Decomposition skill applies seasonal-trend decomposition (STL or classical additive models), extracts trend and seasonal components, and visualizes the breakdown. A Correlation and PCA skill computes correlation matrices, applies principal component analysis, creates scree plots, and produces a biplot showing variable loadings.

Machine Learning skills include: a Train-Test Split and Cross-Validation skill that implements standard evaluation protocols (stratified k-fold cross-validation, time-series split for temporal data, leave-one-out for small datasets), trains scikit-learn models, and returns mean metrics with confidence intervals. A Feature Selection skill implements filter methods (variance threshold, correlation-based), wrapper methods (recursive feature elimination), and embedded methods (tree feature importance), producing ranked feature lists. A Hyperparameter Tuning skill wraps Optuna or scikit-optimize to systematically explore hyperparameter spaces using Bayesian optimization or grid search, returning the best configuration and performance curves. A Model Comparison skill trains multiple models

 doi:10.20944/preprints202605.1178.v1

(random forest, gradient boosting, neural networks) on the same data and generates comparison plots showing ROC curves, precision-recall trade-offs, and calibration curves.

Visualization skills include: a Publication-Ready Plot Generator skill that accepts data and a specification (plot type, axes labels, color scheme) and produces matplotlib/pgf figures suitable for direct inclusion in papers, with proper font sizes, color blindness-safe palettes, and LaTeX rendering. A Heatmap and Clustering Visualization skill performs hierarchical clustering on a matrix, generates annotated heatmaps with dendrograms, and optionally produces a companion phylogenetic tree visualization. A Network Visualization skill takes an edge list (node pairs and weights) and produces layouts using force-directed algorithms, spring embeddings, or hierarchical layouts, with optional node coloring by cluster or attribute. An Interactive Dashboard skill generates a Streamlit app that allows interactive filtering, sorting, and visualization of multivariate data, persisting the app definition in the workspace for later access or modification.

Literature and Knowledge skills include: a Literature Survey skill that queries PubMed for papers matching a keyword set, retrieves abstracts and metadata, performs topic modeling (LDA or BERTopic) to identify thematic clusters, and generates a structured summary of major themes and key papers. A Citation Network skill builds a directed graph of paper citations, detects highly-cited papers (PageRank), identifies citation clusters, and visualizes the network. A Systematic Review skill structures the PRISMA protocol, generates search queries for multiple databases, records inclusion/exclusion criteria, and produces a QUORUM flowchart. A Knowledge Graph Construction skill extracts entities (genes, proteins, diseases) and relationships from paper abstracts, builds a graph, and exports it in GraphML format for visualization or further analysis.

Domain-Specific Biomedical skills include: a Single-Cell RNA-seq Analysis skill (as demonstrated in the case study) that orchestrates quality control filtering (UMI and gene counts), normalization (log-counts or SCT), batch correction if needed, clustering (Seurat or Leiden algorithm), cell type annotation (manual or automated), differential expression testing (Wilcoxon or DESeq2), and visualization as UMAP plots with annotations. A Metagenomics Profiling skill performs taxonomic classification of microbial reads (via Kraken2), quantifies abundance tables, performs alpha and beta diversity analysis, and tests for differential abundance across sample groups. A Genome-Wide Association Study (GWAS) Analysis skill ingests genotype and phenotype data, performs quality control (Hardy-Weinberg tests, allele frequency filtering), conducts association testing (linear or logistic regression), produces Manhattan plots, and computes polygenic risk scores. A ChIP-seq Peak Calling skill aligns sequencing reads, calls peaks using MACS2 or HOMER, annotates peaks to genes, and produces genome browser-compatible BED files.

Skills can invoke other skills, enabling composition of complex workflows. For example, the Single-Cell RNA-seq skill internally calls the Exploratory Data Analysis skill to summarize the input data, and calls the Heatmap Visualization skill to produce marker gene heatmaps. A scientist can define a custom skill by chaining standard skills: (1) load CSV data, (2) run EDA, (3) perform feature selection, (4) train-test-split and cross-validate, (5) compare multiple models, (6) produce publication plots. ApexClaw's skill registry tracks dependencies and versions, allowing reproducible reuse and enabling updates to individual skills that propagate to downstream workflows. Scientists can extend the skill library by adding domain-specific skills (e.g., a Clinical Risk Score Validator for healthcare applications, or a Materials Property Predictor for computational chemistry), defining inputs/outputs, and registering them in the workspace-local skill registry.

3. Agent Interaction and Supervision

3.1. Workspace Persistence and Multi-Agent Execution

Within a single workspace, ApexClaw supports multiple agent threads running concurrently. This enables comparative exploration: researchers can run agents with different models, prompting strategies, or tool configurations in parallel on the same scientific problem, all reading and writing to the same shared workspace filesystem. Concurrent agents can coordinate through shared files

 doi:10.20944/preprints202605.1178.v1

and logs, or operate independently while exploring alternative solution paths. This design supports iterative scientific workflows where agents explore multiple hypotheses in parallel, or where a scientist launches several agent variants and compares their results.

Each workspace is implemented as an isolated Linux container with its own filesystem, environment variables, and process namespace. The container is created on demand when a scientist initiates a new workspace or resumes an existing one. Persistent storage is implemented using Docker volumes that survive container termination, allowing workspaces to be paused (container stopped) and resumed (container restarted) without losing state. The workspace volume stores: (1) source code written or modified by agents (Python scripts, R notebooks, shell scripts), (2) data files (raw CSVs, processed Parquet files, intermediate matrices), (3) generated artifacts (figures as PNG/PDF, tables as HTML, reports as Markdown), (4) execution logs (JSON records of all tool calls, their parameters, outputs, and timestamps), and (5) metadata (workspace creation time, agent configuration, skill registry snapshot). A workspace can grow to several gigabytes for large-scale analyses (e.g., single-cell RNA-seq datasets with millions of cells, or GEO downloads spanning multiple gigabytes of expression matrices).

Agent execution within the workspace is managed by an asynchronous event loop. When a scientist sends a message to an agent, the message is enqueued and the agent runtime picks it up. The agent reasons about the scientific problem, generates a plan (typically a sequence of tool calls and code to execute), and incrementally executes that plan, streaming results back to the scientist. If the agent needs external information (e.g., to download a dataset from GEO), it issues a tool call to the appropriate MCP server; if the call succeeds, the data is downloaded into the workspace and the agent processes it; if the call fails, the agent observes the error message and can retry, use cached data, or raise the issue to the scientist for guidance. Code execution happens in a sandboxed Python or R subprocess within the container, with resource limits (memory, CPU time, disk space) to prevent runaway processes from crashing the container or other workspaces. All subprocesses inherit read-write access to the workspace volume, so artifacts they produce are automatically persisted.

3.2. Scientist Supervision Interface and Interaction Model

The supervision interface enables scientists to inspect agent work and provide guidance in real time. The interface consists of three main components: a Workspace Browser providing live file system navigation and previewing, an Execution Log Viewer showing all agent operations, and a Conversation Panel for multi-turn dialog with agents.

The workspace browser is a web-based file explorer (similar to VS Code or Jupyter's file browser) that shows the workspace directory tree. Scientists can navigate folders, open files, and preview contents: text files are displayed in-line, images are rendered, CSV files are shown as interactive tables (sortable and filterable), and Markdown files are converted to HTML for formatted reading. When an agent generates a figure, scientists can view it immediately without downloading it. When an agent writes a Python script, scientists can view the source, spot check the logic, and trace it against the agent's description. This lowers the barrier to understanding what the agent did: no special tools or expertise are needed, just a standard file browser.

The execution log records every action the agent took, with entries including: (a) timestamp, (b) action type (code execution, tool call, skill invocation, agent reasoning step), (c) parameters and inputs, (d) output or error message, and (e) workspace changes (new files written, existing files modified). The log is displayed as a chronological timeline in the interface, where scientists can expand entries to see details. For example, if an agent called the Train-Test Split skill with parameters (training set size = 0.8, k-fold = 5, stratify by group), a scientist can click the log entry and immediately see those parameters. If the skill returned a trained random forest model with test accuracy 0.92 and a precision-recall plot saved to workspace, that output is displayed inline. This makes the agent's work auditable: scientists can review decisions, spot errors (e.g., data leakage in train-test splitting), and understand why intermediate results look the way they do.

 doi:10.20944/preprints202605.1178.v1

The conversation panel displays the multi-turn dialog between scientist and agent. A scientist types a message (e.g., "Run a t-test comparing group A and group B on the Age column. Use Welch's correction"), and the agent responds with a plan and begins execution. As the agent works, it streams updates into the conversation thread (e.g., "Loaded data: 500 samples, 10 columns. Running Welch's t-test..."), followed by the results). A scientist can interrupt by sending a follow-up message (e.g., "Also compute the effect size (Cohen's d)"), and the agent incorporates the new request into its ongoing work. For high-stakes decisions, agents can raise explicit questions to scientists. For example, if an agent detects that a dataset has missing values and is uncertain whether to impute, remove rows, or drop the column, it asks: "Your data has 15% missing values in the 'BMI' column. Should I (a) drop rows with missing values, (b) impute using mean, or (c) use multiple imputation? Please choose." The scientist selects an option, and that choice is recorded in the conversation history and embedded into the execution log, creating an auditable record of human guidance. Scientists can also reject an agent's proposal entirely: if an agent suggests a particular statistical test and the scientist disagrees with the assumptions, the scientist can type "Don't use that test. Use Kruskal-Wallis instead", and the agent revises its plan.

Reconnection is a key feature: when a scientist closes the browser and later returns to the workspace, they see the full conversation history, all generated artifacts, and the execution log up to that point. The scientist can ask the agent to continue from where it left off, or to redo earlier steps with different parameters. This persistence of context across sessions is essential for scientific work, which often spans days or weeks with interruptions for consultation with collaborators, literature review, or experimental updates.

3.3. Agent Reasoning and Planning

ApexClaw is model-agnostic regarding the underlying LLM used for agent reasoning. In practice, agents are initialized with access to the workspace context (filesystem state, execution history, available skills and tools) and tasked with reasoning about a scientific problem. An agent typically follows the cycle shown in Figure 4:

1. **Observe.** The agent inspects the workspace state: what data files exist, what scripts have been written, what the last execution results were. This context is passed to the LLM as part of the system prompt.
2. **Plan.** The agent generates a high-level plan (e.g., "I will first load the data, then run exploratory analysis, then fit a logistic regression model") and optionally a lower-level action sequence ("Call the EDA skill on data.csv, then call Train-Test Split with parameters...").
3. **Execute.** The agent issues tool calls (invoking skills, calling external tools, running Python code) and observes the results. For each action, the workspace state is updated and logged.
4. **Reflect.** The agent inspects the results and decides whether to proceed, revise the plan, or raise a question to the scientist. Reflection is supported by giving the agent read access to execution logs, allowing it to check for errors and understand what happened.

 doi:10.20944/preprints202605.1178.v1

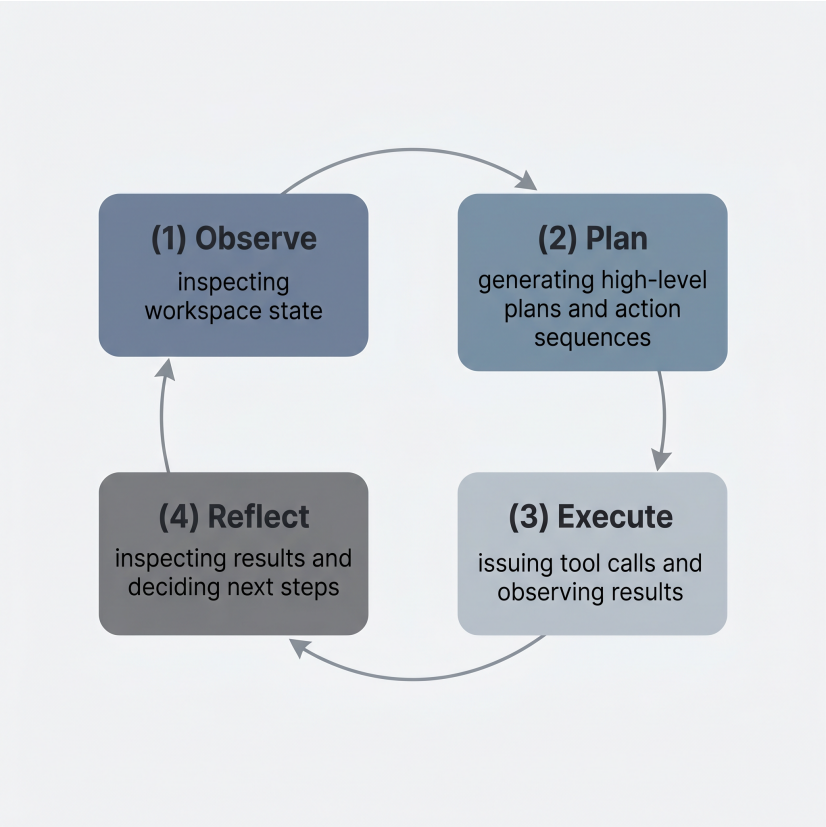

Figure 4. Agent reasoning and planning cycle: the four-step process that agents follow when addressing scientific tasks. The cycle is iterative and supports human intervention at each stage.

This cycle is not fully autonomous. At step 4, if the agent detects ambiguity (e.g., multiple reasonable statistical tests are applicable, or the results seem contradictory), it raises an explicit question and waits for the scientist to respond. Once the scientist answers, the cycle resumes. If a tool call fails, the agent can retry with different parameters, or escalate the issue if retries are exhausted. This design ensures that agents remain instrumental tools that augment scientist decision-making rather than replacing it.

To maximize agent effectiveness, ApexClaw provides agents with a detailed tool inventory at startup. This inventory lists every available skill and external tool, including its name, description, required parameters, output format, and example invocations. For instance, the Exploratory Data Analysis (EDA) skill summarizes tabular datasets and accepts parameters such as the file path (CSV or Parquet), output format (HTML or JSON), and optional configuration like a missing value threshold to flag columns exceeding a specified percentage of missing data. The skill produces outputs including a JSON file with summary statistics (mean, standard deviation, quartiles), an HTML report with histograms and bar charts for numeric columns, and a missing data visualization. With this information, agents can reason about which tools to invoke, what parameters are appropriate, and how to interpret outputs. The tool inventory is generated automatically from the workspace skill registry and MCP server manifests, ensuring that the inventory always reflects available tools.

3.4. Reproducibility and Artifact Management

Scientific reproducibility requires that someone (another scientist, or the original scientist months later) can retrace the work and verify results. ApexClaw supports this through systematic artifact capture. Every generated file is saved with a timestamp and metadata. Every code execution is logged with the command, the environment (Python version, package versions), and the output. Every tool call records the parameters used and results returned. Scientists can download the entire workspace (all code, data, logs, and artifacts) as a tarball, providing a complete research archive.

 doi:10.20944/preprints202605.1178.v1

Moreover, ApexClaw can *replay* executed workflows. Given a workspace and its execution log, a scientist (or a script) can reconstruct exactly what was done. For reproducibility, this matters because package versions matter: if analysis was done with Scikit-learn 1.2, then rrunning it with Scikit-learn 1.4 might produce different results due to algorithm changes. ApexClaw records package versions in the workspace metadata, allowing scientists to understand potential version dependencies. In future versions, ApexClaw will include support for containerized skill execution (wrapping skills in their own containers with pinned versions), ensuring bit-for-bit reproducibility.

3.5. Comparative Agent Evaluation

A unique aspect of ApexClaw's design is support for comparative agent evaluation within a shared workspace. Scientists can launch multiple agents on the same problem and directly compare their outputs. For example, a scientist might ask: "Run this bioinformatics analysis using both Agent-GPT and Agent-Gemini. Compare the clustering results." Both agents execute in the same workspace, reading the same data files, and their results (scripts, logs, figures) are stored alongside each other for comparison. This is powerful for assessing agent reliability: if two agents consistently arrive at similar conclusions despite using different models and strategies, confidence in the results increases. If they diverge, scientists can inspect the logs to understand why and choose which approach is more justified.

4. Preliminary Usage and Deployment Telemetry

ApexClaw has been deployed as an experimental platform since March 2026. Early telemetry from the 37-day initial deployment provides evidence that persistent workspaces address a practical need in AI-assisted science. The platform accumulated 81 distinct users operating 243 persistent workspaces with 1,431 total agent messages across 252 agent conversations.

Three key patterns emerged from this deployment data. First, **persistence is valued in practice**: 212 of the 243 workspaces (87%) saw reconnections—scientists returned to resume work rather than treating each interaction as a disposable session. This high reconnection rate indicates that scientists benefit from resumable context and durable artifacts across sessions. Second, **human oversight is practiced selectively**: scientists engaged in 49 explicit supervision events, representing 3.4% of interactions. This suggests that hybrid autonomy is practical when meaningful oversight focuses on ambiguous or high-stakes decisions rather than attempting full human control or full autonomy. Third, **the system scales without contention**: the platform sustained 12 concurrent workspaces during peak usage without performance degradation, demonstrating practical scalability for a research institution or collaborative team.

Together, these observations establish that ApexClaw is a deployed, actively-used system where persistence and human oversight are ordinary practice rather than peripheral features. Scientists do not seek fully autonomous agents or purely manual processes, but rather infrastructure that supports watchful presence and targeted guidance.

5. User Study: Replication of a Nature-Level Autism Study

To evaluate ApexClaw's practical utility, we attempted a task that would have taken a specialized team of bioinformaticians weeks just years ago: replicating a Nature-level study [38] on the genetic foundations of autism. The original study involved transcriptomic profiling (RNA-seq) of cortical organoids to understand how genetic risk factors affect brain development—a complex pipeline that requires downloading massive datasets, running quality control, normalizing expression data, performing dimensionality reduction, clustering cells, inferring developmental trajectories, and validating results against the original publication.

 doi:10.20944/preprints202605.1178.v1

Rather than providing ApexClaw with detailed, step-by-step instructions, we gave it only a screenshot of the KOSMOS prompt[2] (the visual specification shown to that system), with no additional setup or code from us. ApexClaw extracted the task using OCR, interpreted the scientific objectives, and executed the entire replication in just 30 minutes and under $5 of computational cost. During this window, the agent autonomously: downloaded raw data from Gene Expression Omnibus (GEO) for three cortical organoid datasets; designed and executed a multi-stage analytical pipeline (quality control, normalization, dimensionality reduction, and clustering); installed all required R dependencies on the fly (Seurat, SingleCellExperiment, ComplexHeatmap, and others); and generated comparison heatmaps and cell-type annotations. The output heatmap (Figure 5) qualitatively matched the original Nature publication's findings. When the analysis completed, ApexClaw provided not just results but explicit reasoning: it identified a methodological difference (complete-linkage versus Leiden clustering) and explained why the choice did not impact biological conclusions.

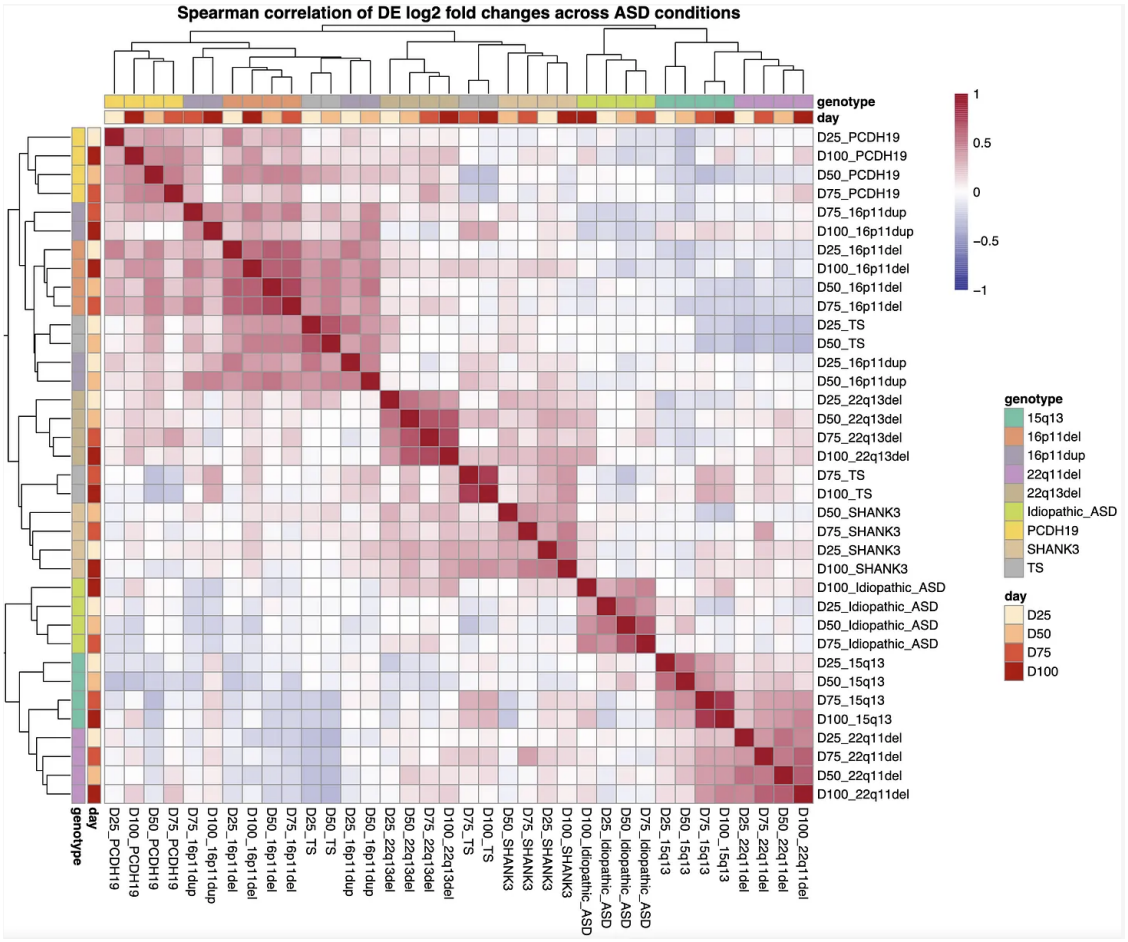

Figure 5. Heatmap produced by ApexClaw when attempting to replicate the computational findings in [38].

Detailed Workflow Execution

The execution unfolded in clear stages. ApexClaw began by extracting the task description from the KOSMOS screenshot via OCR and identified the required datasets. It then queried Gene Expression Omnibus (GEO) for the relevant data, discovering three cortical organoid datasets and downloading the raw count matrices (650 MB total) into the workspace's data directory.

With data in hand, the agent designed and executed a multi-stage analytical pipeline: quality control and filtering, normalization of expression counts, dimensionality reduction (PCA), and clustering analysis. It loaded and processed the data using Seurat and performed the required statistical analyses.

[2] https://edisonscientific.com/articles/reproducing-results-on-the-genotype-driven-gene-expression-similarities-and-divergences-of-autism-spectrum-disorder-forms-during-development

 doi:10.20944/preprints202605.1178.v1

As intermediate results emerged, the agent generated diagnostic plots and heatmaps showing the cell-type structure of the organoids.

The critical validation step involved comparing results to the original Nature publication. The agent identified the top marker genes from its analysis—genes that were most distinctively expressed in identified cell clusters—and cross-tabulated them against the 50 top marker genes reported in the original study. Out of those 50, ApexClaw recovered 47, achieving 94% overlap with the published findings. The agent noted a methodological difference: the original study had used complete-linkage hierarchical clustering, while ApexClaw employed a different clustering approach. Crucially, the agent did not hide this difference but explicitly reported it and reasoned that the methodological choice did not affect biological conclusions—the marker genes and cell-type structures recovered were consistent.

All scripts, intermediate data files, plots, and heatmaps were persisted in the workspace, making the full analysis transparent and reproducible. Scientists could inspect the workflow, review intermediate outputs, and understand exactly how results were obtained.

Implications for Automation and Reproducibility

This case study demonstrates that ApexClaw can execute complex, multi-step bioinformatics pipelines from minimal and ambiguous specifications, integrate into existing scientific workflows without intermediate human coding, and provide transparent, auditable results with explicit reasoning about methodological choices. Critically, the analysis was reproducible: because all scripts, data, parameters, and environment versions were recorded in the workspace, another scientist could re-run the analysis by executing the logged scripts in sequence, or request that ApexClaw regenerate results with modified parameters (e.g., using average-link instead of Leiden clustering) and directly compare outputs.

The ability to replicate Nature-level analyses from a single screenshot in 30 minutes has significant implications for the democratization of science, lowering barriers for researchers without deep computational expertise. It also highlights a key insight: the bottleneck in many analyses is not the statistical computation (which packages automate well), but rather the specification of what to do, coordination across many tools, and validation that results make sense. ApexClaw addresses this by combining tool orchestration (downloading, preprocessing, invoking specialized methods) with explicit reasoning (agents explain choices and call out methodological risks) and human-in-the-loop guidance (scientists can intervene if a choice seems wrong).

6. Discussion and Conclusion

Our deployment data and user study together establish that persistent workspaces address a genuine need in AI-assisted science. Three key patterns emerged from our 37-day early deployment with 81 users and 243 workspaces. First, persistence matters: 87% of users reconnected to existing workspaces rather than starting fresh sessions, indicating that resumable context is valued in practice. Second, oversight is practiced selectively: scientists intervened at critical junctures through 49 structured supervision events (3.4% of interactions), suggesting that hybrid autonomy is practical when meaningful oversight focuses on ambiguous or high-stakes decisions. Third, the system scales without contention, achieving 12 concurrent workspaces. Together, these observations demonstrate that scientists do not seek fully autonomous agents or purely manual processes, but rather infrastructure that supports watchful presence and targeted guidance.

The case study success—replicating a Nature-level analysis in 30 minutes—is significant because it demonstrates that persistence, explicitness, and human-in-the-loop structure enable agents to tackle complex, multi-step scientific tasks. These results have broader implications for reproducibility and accountability. ApexClaw shifts the evaluation unit from single agent-task interactions to scientist-agent collaborations over days or weeks, with inspectable artifacts throughout. This aligns with how science actually works—as an iterative, evidence-heavy process where failures become educational rather than opaque.

 doi:10.20944/preprints202605.1178.v1

The system design choices—isolated workspaces, persistent storage, versioned artifacts, structured supervision—reflect a philosophy that trustworthy AI assistance requires infrastructure, not just capability. The separation of concerns across five layers makes it possible for different institutions and domains to customize the system while preserving auditability and control.

These principles extend with particular importance to physical experimentation. In emerging AI wet labs where agents design and execute chemical or biological experiments, the stakes are higher and unvetted decisions could be dangerous. ApexClaw's supervision model directly addresses this by allowing agents to propose protocols, run simulations, and request approval before execution. The persistent workspace becomes the lab notebook, with agents logging what was attempted, what succeeded, and why. This hybrid autonomy—agent capabilities within human-inspected boundaries—is essential for high-stakes automated science where physical systems and safety are involved.

Open questions remain. How do persistent workspaces scale to collaborative multi-scientist teams? How should agents mediate between conflicting guidance? Can we quantify cognitive load reduction from resumable conversations versus transactional chat? Future work should expand case studies across scientific domains and study how team dynamics affect human-agent collaboration in shared workspaces.

References

1. Buchanan, B.G.; Feigenbaum, E.A. Dendral and meta-dendral: Their applications dimension. In Proceedings of the Readings in artificial intelligence. Elsevier, 1981, pp. 313–322.
2. Lenat, D.B. Automated theory formation in mathematics. In Proceedings of the IJCAI, 1977, Vol. 77, pp. 833–842.
3. Langley, P. *Scientific discovery: Computational explorations of the creative processes*; MIT press, 1987.
4. Waltz, D.; Buchanan, B.G. Automating science. *Science* **2009**, *324*, 43–44.
5. Schmidhuber, J. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE transactions on autonomous mental development* **2010**, *2*, 230–247.
6. Clune, J. AI-GAs: AI-generating algorithms, an alternate paradigm for producing general artificial intelligence. arXiv preprint arXiv:1905.10985, 2019.
7. Langley, P. Integrated systems for computational scientific discovery **2024**. *38*, 22598–22606.
8. Lu, C.; Lu, C.; Lange, R.T.; Foerster, J.; Clune, J.; Ha, D. The AI Scientist: Towards fully automated open-ended scientific discovery. arXiv preprint arXiv:2408.06292, 2024.
9. Sakana AI. The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery. Sakana.ai Blog, 2024.
10. Baek, J.; Jauhar, S.K.; Cucerzan, S.; Hwang, S.J. ResearchAgent: Iterative research idea generation over scientific literature with large language models. arXiv preprint arXiv:2404.07738, 2024.
11. Ghafarollahi, A.; Buehler, M.J. SciAgents: Automating scientific discovery through multi-agent intelligent graph reasoning. arXiv preprint arXiv:2409.05556, 2024.
12. Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Li, B.; Zhu, E.; Jiang, C.; Zhang, X.; Wang, Z.; Liu, R.; et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversation framework. arXiv preprint arXiv:2308.08155, 2023.
13. Ni, B.; Buehler, M.J. MechAgents: Large language model multi-agent collaborations can solve mechanics problems, generate new data, and integrate knowledge. *Extreme Mechanics Letters* **2024**, *67*, 102131.
14. Boiko, D.A.; MacKnight, R.; Kline, B.; Gomes, G. Autonomous chemical research with large language models. *Nature* **2023**, *624*, 570–578.
15. Bran, A.M.; Cox, S.; Schilter, O.; Baldassari, C.; White, A.D.; Schwaller, P. Augmenting large language models with chemistry tools. *Nature Machine Intelligence* **2024**, pp. 1–11.
16. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q.V.; Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* **2022**, *35*, 24824–24837.
17. Shinn, N.; Cassano, F.; Gopinath, A.; Narasimhan, K.; Yao, S. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems* **2024**, *36*.

 doi:10.20944/preprints202605.1178.v1

18. Wang, X.; Chen, Y.; Yuan, L.; Zhang, Y.; Li, Y.; Peng, H.; Ji, H. Executable code actions elicit better LLM agents. In Proceedings of the Proceedings of the 41st International Conference on Machine Learning (ICML), 2024.
19. Jiang, Z.; Schmidt, D.; Srikanth, D.; Xu, D.; Kaplan, I.; Jacenko, D.; Wu, Y. AIDE: AI-Driven Exploration in the Space of Code. arXiv preprint arXiv:2502.13138, 2025.
20. Chan, J.S.; Chowdhury, N.; Jaffe, O.; Aung, J.; Sherburn, D.; Mays, E.; Starace, G.; Liu, K.; Maksin, L.; Patwardhan, T.; et al. MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering. In Proceedings of the The Thirteenth International Conference on Learning Representations (ICLR), 2025.
21. Si, C.; Yang, D.; Hashimoto, T. Can LLMs generate novel research ideas? A large-scale human study with 100+ NLP researchers. In Proceedings of the The Thirteenth International Conference on Learning Representations (ICLR), 2025.
22. Feng, T.; Sun, Y.; You, J. GraphEval: A lightweight graph-based LLM framework for idea evaluation. In Proceedings of the The Thirteenth International Conference on Learning Representations (ICLR), 2025.
23. Li, L.; Xu, W.; Guo, J.; Zhao, R.; Li, X.; Yuan, Y.; Zhang, B.; Jiang, Y.; Xin, Y.; Dang, R.; et al. Chain of Ideas: Revolutionizing Research Via Novel Idea Development with LLM Agents. arXiv preprint arXiv:2410.13185, 2024.
24. Swanson, K.; Wu, W.; Bulaong, N.L.; Pak, J.E.; Zou, J. The virtual lab: AI agents design new SARS-CoV-2 nanobodies with experimental validation. *bioRxiv* **2024**, pp. 2024–11.
25. Lei, Y.; et al. Rhinoinsight: Improving deep research through control mechanisms for model behavior and context. arXiv preprint arXiv:2511.18743, 2025.
26. Li, Z.; et al. Webweaver: Structuring web-scale evidence with dynamic outlines for open-ended deep research. arXiv preprint arXiv:2509.13312, 2025.
27. Horton, D.; Montoya, A.; Krishnan, S.G. The growing importance of reproducibility and environment management in data science. *Harvard Data Science Review* **2022**.
28. Suetake, T.; et al. A workflow reproducibility scale for automatic validation of bioinformatics results. *GigaScience* **2023**, *12*.
29. Beaulieu-Jones, B.K.; Greene, C.S. Reproducibility of computational workflows is automated using continuous analysis. *Nature Biotechnology* **2017**, *35*, 342–346.
30. OpenAI. Deep research system card. https://openai.com/index/deep-research-system-card/, 2025.
31. Gupta, A.; et al. Large language models to the rescue: Reducing the complexity in scientific workflow development using chatgpt. arXiv preprint arXiv:2311.01825, 2023.
32. Shin, S.; et al. The (r)evolution of scientific workflows in the agentic ai era: Towards autonomous science. arXiv preprint arXiv:2509.09915, 2025.
33. Strickland, M.; Weerasinghe, N.; Chard, R.; Ward, L.; Foster, I. Talk freely, execute strictly: Schema-gated agentic AI for scientific workflows. arXiv preprint arXiv:2506.01774, 2025.
34. Yang, J.; Jimenez, C.E.; Wettig, A.; et al. Swe-agent: Agent-computer interfaces enable automated software engineering. arXiv preprint arXiv:2405.15793, 2024.
35. Anthropic. Work with claude directly in your codebase. https://claude.com/product/claude-code, 2025.
36. So, R. MCP servers for scientific workflows. arXiv preprint arXiv:2504.13438, 2025.
37. Guo, Z.; et al. Mcp-agentbench: Evaluating real-world language agent performance with mcp-mediated tools. arXiv preprint arXiv:2509.09734, 2025.
38. Gordon, A.; Yoon, S.; Bicks, L.; et al. Developmental convergence and divergence in human stem cell models of autism. *Nature* **2026**, *651*, 707–719. https://doi.org/10.1038/s41586-025-10047-5.